📘 Compiler Design – UNIT III

🔥 Type Checking & Run Time Environment

🎯 Easy + Detailed + RGPV Exam Writing Style

🌟 UNIT III SYLLABUS

📌 Type Checking

- Type System
- Specification of Simple Type Checker
- Equivalence of Types
- Type Conversion
- Function & Operator Overloading
- Polymorphic Functions

📌 Run Time Environment

- Storage Organization
- Storage Allocation Strategies
- Parameter Passing
- Dynamic Storage Allocation
- Symbol Table

🔥 MOST IMPORTANT 14-MARK QUESTIONS

1. Explain Type Checking.
2. Explain Type System.
3. Explain Type Conversion.
4. Explain Function and Operator Overloading.
5. Explain Polymorphic Functions.
6. Explain Storage Organization.
7. Explain Storage Allocation Strategies.
8. Explain Parameter Passing Methods.
9. Explain Dynamic Storage Allocation.
10. Explain Symbol Table.

🔥 Q1. Explain Type Checking.

✍️ Answer:

📌 Definition

Type Checking is the process of verifying whether operands and operators are used with compatible data types.

🌟 Purpose of Type Checking

✅ Detect type errors
✅ Ensure program correctness
✅ Prevent invalid operations

🌟 Example

Correct:

```
int a = 10;
```

Incorrect:

```
int a = "Hello";
```

This produces:

```
Type Mismatch Error
```

🌟 Types of Type Checking

Type	Description
Static Type Checking	Performed during compilation
Dynamic Type Checking	Performed during execution

🌟 Advantages

✅ Improves reliability
✅ Detects errors early
✅ Reduces runtime errors

📌 Conclusion

Type Checking ensures that operations are performed on compatible data types.

🔥 Q2. Explain Type System.

✍️ Answer:

📌 Definition

A Type System is a set of rules used to define and check data types in a programming language.

🌟 Types of Data Types

Data Type	Example
Integer	int
Floating Point	float
Character	char
Boolean	true/false

🌟 Functions of Type System

✅ Defines valid operations

✅ Detects type errors

✅ Improves program safety

🌟 Example

```
int a = 10;
float b = 5.5;
```

Addition:

```
a + b
```

Allowed because type conversion is possible.

🌟 Advantages

✅ Program correctness
✅ Better readability
✅ Error detection

📌 Conclusion

Type System controls the usage of data types and ensures safe operations.

🔥 Q3. Explain Specification of Simple Type Checker.

✍️ Answer:

📌 Definition

A Simple Type Checker verifies the types of variables and expressions according to language rules.

🌟 Working of Type Checker

```
Source Program
      ↓
Type Checker
      ↓
Type Verified Program
```

🌟 Example

```
int a;
float b;

a = b;
```

Type checker verifies whether assignment is valid.

🌟 Functions

✅ Checks assignment compatibility
✅ Checks expression types
✅ Detects invalid operations

📌 Conclusion

A Simple Type Checker ensures proper use of data types in a program.

🔥 Q4. Explain Equivalence of Types.

✍️ Answer:

📌 Definition

Type Equivalence determines whether two data types are considered identical.

🌟 Types of Type Equivalence

Type	Description
Name Equivalence	Types are same if names are same
Structural Equivalence	Types are same if structures are same

🌟 Example

```
int a;
int b;
```

Both are equivalent types.

🌟 Advantages

✅ Simplifies type checking

✅ Ensures compatibility

📌 Conclusion

Type Equivalence is used to compare data types during compilation.

🔥 Q5. Explain Type Conversion.

✍️ Answer:

📌 Definition

Type Conversion means converting one data type into another.

🌟 Types of Type Conversion

Type	Description
Implicit Conversion	Automatic conversion
Explicit Conversion	Manual conversion

🌟 Example

Implicit:

```
int a = 10;
float b = a;
```

Explicit:

```
float x = 5.6;
int y = (int)x;
```

🌟 Advantages

✅ Allows mixed type operations
✅ Improves flexibility

🌟 Disadvantages

❌ Data loss may occur

📌 Conclusion

Type Conversion helps in performing operations between different data types.

🔥 Q6. Explain Function and Operator Overloading.

✍️ Answer:

🌟 Function Overloading

📌 Definition

Using the same function name with different parameter lists.

🌟 Example

```
int add(int a,int b);

float add(float a,float b);
```

🌟 Advantages

✅ Code reusability
✅ Better readability

🌟 Operator Overloading

📌 Definition

Giving special meaning to operators for user-defined data types.

🌟 Example

```
obj1 + obj2
```

🌟 Advantages

✅ Simplifies programming

✅ Makes code user friendly

📌 Conclusion

Overloading improves flexibility and readability in programming.

🔥 Q7. Explain Polymorphic Functions.

✍️ Answer:

📌 Definition

Polymorphic Functions can work with multiple data types.

🌟 Example

```
template <class T>
T add(T a,T b)
{
   return a+b;
}
```

🌟 Advantages

✅ Code reusability
✅ Flexibility
✅ Reduces duplication

📌 Conclusion

Polymorphic Functions support multiple data types using a single function.

🔥 Q8. Explain Run Time Environment.

✍️ Answer:

📌 Definition

Run Time Environment is the memory and data management system used during program execution.

🌟 Functions

✅ Memory management

✅ Parameter handling

✅ Dynamic allocation

✅ Procedure calls

🌟 Components

Component	Function
Stack	Function calls
Heap	Dynamic memory
Code Area	Program instructions
Data Area	Global variables

📌 Conclusion

Run Time Environment manages resources required during program execution.

🔥 Q9. Explain Storage Organization.

✍️ Answer:

📌 Definition

Storage Organization refers to memory arrangement during execution.

🌟 Memory Layout

```
Code Area
↓
Global Data
↓
Heap
↓
Stack
```

🌟 Components

1. Code Area

Stores program instructions.

---

2. Data Area

Stores global/static variables.

---

3. Heap

Stores dynamically allocated memory.

---

4. Stack

Stores function calls and local variables.

📌 Conclusion

Storage Organization helps efficient memory management during execution.

🔥 Q10. Explain Storage Allocation Strategies.

✍️ Answer:

📌 Definition

Storage Allocation Strategy determines how memory is allocated to variables.

🌟 Types

Type	Description
Static Allocation	Memory fixed before execution
Stack Allocation	Memory allocated during function calls
Heap Allocation	Dynamic memory allocation

🌟 Static Allocation

Memory assigned before execution.

Example:

```
int a;
```

🌟 Stack Allocation

Memory allocated during function calls.

🌟 Heap Allocation

Dynamic allocation using:

```
malloc()
new
```

📌 Conclusion

Different allocation strategies are used for efficient memory usage.

🔥 Q11. Explain Parameter Passing Methods.

✍️ Answer:

📌 Definition

Parameter Passing transfers data between functions.

🌟 Types

Method	Description
Call by Value	Copy of variable passed
Call by Reference	Original variable passed

🌟 Call by Value

```
void fun(int x)
```

Changes do not affect original variable.

🌟 Call by Reference

```
void fun(int &x)
```

Changes affect original variable.

📌 Conclusion

Parameter Passing methods control how arguments are transferred to functions.

🔥 Q12. Explain Dynamic Storage Allocation.

✍️ Answer:

📌 Definition

Dynamic Storage Allocation allocates memory during execution.

🌟 Example

```
int *p;
p = (int*)malloc(sizeof(int));
```

🌟 Advantages

✅ Efficient memory usage

✅ Flexible allocation

🌟 Disadvantages

❌ Memory leakage possible

📌 Conclusion

Dynamic Storage Allocation provides flexible memory management during runtime.

🔥 Q13. Explain Symbol Table.

✍️ Answer:

📌 Definition

Symbol Table is a data structure used by compiler to store information about identifiers.

🌟 Stores Information

Identifier	Type	Address
a	int	2000

🌟 Functions

✅ Stores variable names

✅ Stores data types

✅ Stores scope information

🌟 Advantages

✅ Fast identifier lookup

✅ Efficient compilation

📌 Conclusion

Symbol Table plays an important role in compiler design and semantic analysis.

⚡ ONE NIGHT REVISION

✔ Type Checking verifies data types
✔ Type System defines type rules
✔ Type Conversion = one type to another
✔ Overloading uses same name differently
✔ Polymorphism supports multiple types
✔ Stack stores function calls
✔ Heap stores dynamic memory
✔ Static, Stack and Heap allocation are important
✔ Symbol Table stores identifiers
✔ Call by Value uses copy
✔ Call by Reference uses original variable